Java ►►► LENGUAJE DE PROGRAMACIÓN

PC WORLD PERÚ

Java surge desde Sun Microsystems en 1991 como un lenguaje de programación sencillo y universal destinado a electrodomésticos que, al desarrollar un código "neutro", no dependía del tipo de máquina en el que se iba a ejecutar.



► **Java se ejecuta** sobre una "máquina virtual", denominada Java Virtual Machine (JVM) que interpretaba el código neutro (independiente del procesador). A finales de 1995, Java se reconvierte en un lenguaje de programación utilizable en Internet produciendo una verdadera revolución en el mundo de las computadoras. La promesa de Java consiste en llegar a ser el "nexo universal" que conecte a los usuarios con la información, esté donde esté, bajo la premisa "Write once, run everywhere".

► **Java 1.1 aparece** a principios de 1997 mejorando mucho la primera versión del lenguaje. Java 1.2 (Java 2) aparece a finales de 1998 incorporando nuevos elementos. Según sus creadores, Sun Microsystems, esta es la primera versión realmente profesional. Hoy en día, el lenguaje Java está siendo utilizado en diferentes dispositivos móviles como celulares, PDA, etc.

► **En esta ficha práctica** se presentará las funciones, variables, sentencias y demás componentes del lenguaje en sí; sin embargo, existen diferentes interfases gráficas que nos permitirán programar en Java con mayor comodidad, y que entregaremos más adelante.

►►► ANTES DE EMPEZAR A PROGRAMAR

Java posee algunas características especiales que valdría la pena recordar antes de empezar a programar:

► En java, cualquier código escrito debe estar dentro del contexto de una clase. La ejecución de un programa comienza por la clase que posee el método *main*, a esta clase se le suele llamar "*main class*". Si no existe un *main class*, el programa arrojará un error al momento de ejecutarse. El resto de clases (de modo muy básico), funcionan como entornos que nos ayudan a crear objetos con funcionalidad (métodos) y características (variables miembro).

► Para compilar una clase desde *Consola,* se utiliza el comando *javac*. Esta utilidad se encuentra en la ruta *%\j2sdk*\bin*; sin embargo, es posible configurar el entorno de variables para que el sistema operativo sepa donde buscarla. Para ejecutar una clase utilizamos el comando *java*.

► En Java es obligatorio que el nombre del archivo que contenga una clase tenga el mismo nombre que la clase. Recordar que Java es sensible a mayúsculas y minúsculas.

►►► COMENTARIOS

► Un comentario se utiliza para colocar información en código fuente sobre alguna parte específica de este; ya sea información sobre clases, su comportamiento, métodos, variables, o de cualquier proceso en general. Existen diferentes maneras de realizar comentarios en Java. Estas son:

►► **// Comentario de una línea:** Permite hacer comentarios de una sola línea y evita comentar inadvertidamente varias líneas de código.

►► **/* Comentario de varias líneas */:** Permite hacer comentarios que ocupen varias líneas.

►► **/** Comentario de documentación */:** Permite hacer comentarios que luego se incluirán en la documentación que la utilidad *javadoc* genera automáticamente.

CARACTERÍSTICAS DE JAVA

Las principales características que ofrece Java frente a otro lenguaje de programación, son:

- **Simple:** Java ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de estos. Java se diseñó para ser parecido a C++ y así facilitar un aprendizaje rápido y fácil, aunque añade características útiles como el *garbage collector* (reciclador de memoria dinámica).
- **Orientado a objetos:** Java trabaja con sus datos como objetos y con interfases a esos objetos. Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo.
- **Distribuido:** Java se ha construido con extensas capacidades de interconexión TCP/IP. Existen librerías de rutinas para acceder e interactuar con protocolos como http y ftp. Esto permite a los programadores acceder a la información a través de la red con tanta facilidad como a los archivos locales. Proporciona las librerías y herramientas para que los programas puedan ser distribuidos, es decir, que se corran en varias máquinas, interactuando.
- **Robusto:** Java realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos en Java ayuda a detectar errores, lo antes posible, en el ciclo de desarrollo. Java obliga a la declaración explícita de métodos, reduciendo así las posibilidades de error. Implementa los arrays auténticos, en vez de listas enlazadas de punteros, con comprobación de límites, para evitar la posibilidad de sobreescribir o corromper memoria, resultado de punteros que señalan a zonas equivocadas. Además, para asegurar el funcionamiento de la aplicación, realiza una verificación de los byte-codes, que son el resultado de la compilación de un programa Java.
- **Arquitectura neutral:** El compilador Java compila su código a un archivo objeto de formato independiente de la arquitectura de la máquina en que se ejecutará. Cualquier máquina que tenga el sistema de ejecución (run-time) puede ejecutar ese código objeto, sin importar la máquina en que ha sido generado.
- **Seguro:** El código Java pasa muchos tests antes de ejecutarse en una máquina. El código se pasa a través de un verificador de byte-codes que comprueba el formato de los fragmentos de código y aplica un probador de teoremas para detectar fragmentos de código ilegal. El *Cargador de Clases* también ayuda a Java a mantener su seguridad, separando el espacio de nombres del sistema de archivos local, del de los recursos procedentes de la red.
- **Portable:** Más allá de la portabilidad básica por ser de arquitectura independiente, Java implementa otros estándares de portabilidad para facilitar el desarrollo. Los enteros son siempre enteros y además, enteros de 32 bits en complemento a 2. Adicionalmente, Java construye sus interfases de usuario a través de un sistema abstracto de ventanas de forma que estas puedan ser implantadas en entornos Unix, PC o Mac.
- **Interpretado:** El intérprete Java (sistema run-time) puede ejecutar directamente el código objeto. Enlazar un programa, normalmente, consume menos recursos que compilarlo, por lo que los desarrolladores con Java pasarán más tiempo desarrollando y menos esperando por la computadora.
- **Multithreaded:** Al ser multithreaded, Java permite muchas actividades simultáneas en un programa. Los *threads* (a veces llamados, procesos ligeros), son básicamente pequeños procesos o piezas independientes de un gran proceso. Al estar los *threads* contruidos en el lenguaje, son más fáciles de usar y más robustos que sus homólogos en C o C++.
- **Dinamico:** Java se beneficia todo lo posible de la tecnología orientada a objetos. Java no intenta conectar todos los módulos que comprenden una aplicación hasta el tiempo de ejecución. Las librerías nuevas o actualizadas no paralizarán las aplicaciones actuales (siempre que mantengan el API anterior)., y simplifica también el uso de protocolos nuevos o actualizados.

TRABAJANDO CON LOS DATOS

- **Tipos de datos primitivos:** Los tipos de datos primitivos almacenan valores que se interpretan directamente, a diferencia de los tipos de datos que el programador define.

Tipo	Bytes	Valor inicial
boolean	true o false	False
char	Carácter de 16 bits	"\u0000"
string	Cadena de caracteres.	" "
byte	Entero de 8 bits con signo	0
short	Entero de 16 bits con signo	0
int	Entero de 32 bits con signo	0
long	Entero de 64 bits con signo	0
float	Número flotante de 32 bits	+0.0f
double	Número flotante de 64 bits	+0.0f

- **Variables:** Son espacios de memoria que serán utilizados para almacenar datos de algún tipo. Pueden cambiar su valor a lo largo del programa pero no su tipo. Su declaración –en el estilo TipoDeDato Nombre-- puede incluir inicialización: Tipo Nombre = valor

char miNombre;
int mimes = 3;

Dentro de las clases, a las variables miembro se les denominan también con el nombre de campos. Este tipo de variables existen, conceptualmente, para almacenar datos característicos de los objetos que sean instancias de alguna clase. Vale aclarar, sin embargo, que también es posible declarar variables locales a un método específico de la clase. A diferencia de las anteriores, estas variables existirían únicamente dentro del contexto del método.

Si no deseamos que la variable se asocie con instancias de la clase, esta se declara como *static*. Luego, existirá solo una copia de esa variable (o campo) independientemente del número de instancias de la clase.

- **Casting:** Casting es pasar el valor de una variable de un tipo a otra variable de otro tipo distinto. En Java solamente se pueden realizar *castings* que tengan sentido; por ejemplo entre un *float* y un *int*. Del siguiente modo: **int entero = (int) flotante; //** en donde flotante es una variable con un valor de tipo flota.

- **Constantes:** Son espacios de memoria al igual que las variables. La diferencia entre ellas radica en que una constante no puede cambiar ni su tipo ni su valor en ninguna parte del programa. Para hacer constante un valor asignado a un identificador, declaramos la variable como final. En el siguiente ejemplo, mostramos además como pueden agruparse constantes relacionadas dentro de una clase:

```
class Evaluacion {
        final static int NotaMinima = 0;
        final static int NotaAprobatoria = 11;
        final static int NotaMaxima = 20;
}
```

Con esta declaración es perfectamente válido acceder a los valores como *objEvaluacion•NotaAprobatoria, objEvaluacion•NotaMaxima*, etc. Suponiendo que *objEvaluación* es un objeto instanciado de la clase *Evaluacion*.

▸▸▸ PROGRAMACIÓN EN JAVA

Cuando se programa en Java, se coloca todo el código en métodos, de la misma forma que se escriben funciones en lenguajes como C. Así tenemos, además de poder poner los comentarios explicados en la carátula, lo siguiente:

▸ **Identificadores:** Estos son nombres que se colocan a cualquier elemento que el programador necesite usar; como por ejemplo, variables, funciones, clases y objetos. Un identificador comienza con una letra o el carácter de subrayado (_). Los siguientes caracteres pueden ser letras o dígitos. Se distinguen las mayúsculas de las minúsculas, y no hay longitud máxima.
Ejemplos son: **miIdentificador, _miIdentificador, mi_Identificador**

▸ **Separadores:** Definirán la forma y función del código. Los separadores admitidos en Java son:

Separador	Función
( ) paréntesis	Para contener listas de parámetros en la definición y llamada a métodos. También se utiliza para definir precedencia en expresiones, contener expresiones para control de flujo y rodear las conversiones de tipo.
{ } llaves	Para contener los valores de matrices inicializadas automáticamente. También se utiliza para definir un bloque de código, para clases, métodos y ámbitos locales.
[ ] corchetes	Para declarar tipos matriz. También se utiliza cuando se referencian valores de matriz.
; punto y coma	Separa sentencias. También se utiliza para encadenar sentencias dentro de una sentencia for.
, coma	Separa identificadores consecutivos en una declaración de variables. También se utiliza para separar parámetros pasados o recibidos a métodos.
. punto	Para separar nombres de paquete, de subpaquetes y clases. También se utiliza para separar una variable o método de una variable de referencia.

▸ **Códigos de Escape:** Son caracteres que pueden ser usados en un contexto de cadenas de caracteres –o String--, para modificar el formato de salida de un texto. Por ejemplo, una sentencia como la siguiente: **System.out.println("Registro personal\n\tIngresa tu nombre:");** en la salida estándar se vería así:
Registro personal
Ingresa tu nombre:
Los códigos de escape soportados son:

Código	Descripción
\n	Nueva línea.
\r	Retorno de carro.
\t	Tabulador horizontal.
\v	Tabulador vertical.
\b	Espacio hacia atrás.
\a	Alerta.
\f	Avance de página.
\\	Barra hacia atrás.
\?	Interrogación.
\'	Comilla simple.
\"	Comilla doble.
\0	El entero 0.

▸ **Operadores aritméticos:** En orden de prioridad, estos son:

Operador	Descripción	Operador	Descripción
+m	+ unario	m % n	Módulo o Resto
-m	- unario	m + n	Suma
m * n	Multiplicación	m - n	Resta
m / n	División		

▸ **Operadores de Incremento y Decremento:** Al igual que en C/C++ representan expresiones abreviadas, a continuación se muestran su equivalencias:

Operador	Descripción	Operador	Descripción
m++	Nueva línea.	m--	Avance de página
++m	Retorno de carro.	--m	Barra hacia atrás.

Los operadores con el prefijo *Pre*, realizan primero la operación sobre su variable asociada para retornar luego el nuevo valor; mientras que los operadores con prefijo *Post*, retornan primero el valor y luego realizan la operación sobre su variable.

▸ **Operadores relacionales:** Se utilizan para comparar dos expresiones: Por ejemplo, **expresión_1 operador expresión_2**

Operador	Descripción	Operador	Descripción
==	Igual	<=	Menor o igual que
!=	No igual	>	Mayor que
<	Menor que	>=	Mayor o igual que

Aunque Java y C++ soportan los mismos operadores relacionales, devuelven valores diferentes. Los operadores relacionales en Java devuelven un tipo booleano, true o false (según se cumpla o no la realción); mientras que en C++ devuelven un valor entero, en donde se puede considerar al valor cero como false, y a un valor no-cero como true.

▸ **Operadores lógicos:** Utilizados para construir expresiones lógicas. Una expresión lógica, sin importar cuan compleja sea, devolverá finalmente un valor de verdadero (true) o falso (false). La siguiente es una expresión lógica típica: **if(nota<=20 && nota>=0)**

Operador	Descripción
!	Negación lógica
&&	Y lógico
||	O lógico

Si el operando a la izquierda de && es falso, o el de || es verdadero, el operando de la derecha ya no se evalúa (Evaluación de cortocircuito).

▸ **Operadores de asignación:** Son los operadores más comunes, y son utilizados para asignar valores a una o más variables. **a = 12.5**; El operador es asociativo por derecha.
x = y = z = 122; En Java existen además los siguientes operadores de asignación:

Operador	Descripción	Descripción
+=	m += n	M = m + n
-=	m -= n	M = m - n
*=	m *= n	M = m * n
/=	m /= n	M = m / n
%=	m %= n	M = m % n
>>=	m >>= n	M = m >> n
<<=	m <<= n	M = m << n
&=	m &= n	M = m & n
^=	m ^= n	M = m ^ n
|=	m |= n	M = m | n

ARREGLOS O ARRAYS

Un *array* es una colección de elementos del mismo tipo. Java, a diferencia de C++, dispone de un tipo array. En C++, un array es simplemente un conjunto de posiciones secuenciales de memoria a las que se puede acceder mediante un índice o punteros. Esto no implica control alguno sobre ese conjunto de posiciones de memoria. En Java, al ser un tipo de datos verdadero, se dispone de comprobaciones exhaustivas del correcto manejo del array; por ejemplo, de la comprobación de sobrepasar los límites definidos para el array, en evitación de desbordamiento o corrupción de memoria.

En Java hay que declarar un array antes de poder utilizarlo. Y en la declaración hay que incluir el nombre del array y el tipo de datos que se van a almacenar en él.

Tipo[] NombreArray = new Tipo[tamaño]
Pueden construirse arrays de arrays:

int tabla[][] = new int[4][5];

CADENAS O STRINGS

Una secuencia de datos de tipo carácter se conoce como string (cadena), y en el entorno Java está implementada por la clase *String* (detalles de esta clase más adelante).

String miCadena = "una cadena cualquiera";
Java cuenta con una herramienta realmente potente al permitirnos concatenar variables *String* utilizando simplemente un signo "+".

Tipo	Descripción
indexOf(cadena)	Primera posición de cadena.
indexOf(cadena, inicio)	Primera posición de cadena a partir de inicio.
lastIndexOf(cadena)	Ultima posición de cadena.
lastIndexOf(cadena, inicio)	Ultima posición de cadena antes de inicio
equals(otroString)	Compara dos cadenas de texto, en longitud y contenido.
equalsIgnoreCase(otroString)	Compara dos cadenas de texto, en longitud y contenido, ignorando diferencias entre mayúsculas y minúsculas.
compareTo(otroString)	Devuelve un entero que indica el "orden" de la cadena original con respecto a la cadena *otroString*.
compareTo(otroString)	Devuelve un entero que indica el "orden" de la cadena original con respecto a la cadena *otroString*.

Tipo	Descripción
startsWith(prefijo,inicio)	Devuelve true si la cadena comienza (en la posición inicio) con el prefijo indicado.
endsWith(sufijo)	Devuelve true si la cadena finaliza con el sufijo indicado.
regionMatches(inicio, otroString, otroInicio, cuenta) regionMatches(ignoraMayusculas, inicio, otroString, otroInicio, cuenta)	Comprueba si la región indicada en la cadena actual (desde inicio hasta inicio+cuenta) existe en la región especificada en la cadena otroString (desde otroInicio hasta otroInicio+cuenta).

MÉTODOS

Son bloques de código que encierran un proceso, pueden o no retornar un valor al lugar donde fueron invocados. Suelen ser comparados con las funciones en C++; sin embargo, es importante mantener en mente las diferencias. Un método representa una funcionalidad de un objeto, mientras que una función en C++ es simplemente un proceso, no necesariamente ligado a alguna variable (en este caso, un objeto).

▸ **Definición de métodos:** Es la implementación de un método; es decir, la definición del proceso que realizará. En su forma más general, la cabecera de un método se vería así:

[especificadordeAcceso] [modificadores] tipo nombreMetodo (tipo1 argumento1, ...) [throws listadeExcepciones]

```
{
...
Bloque de Sentencias;
...
}
```

Se puede usar *return* para indicar el valor que debe devolver la función, de la siguiente manera: **return(expresión).**
El tipo indica el valor de retorno del método. Si no retorna ningún valor se utiliza la palabra clave *void*.

Los modificadores definen algunas características del método.
El *especificadordeAcceso* define el nivel de acceso del entorno hacia el método (también se utiliza para las variables miembro).

Modificadores de un método:	
Abstract	No se define su cuerpo en la clase.
Static	Ya comentado
Final	Impide que una subclase modifique el método
Synchronized	No se puede acceder al mismo tiempo a dicho método desde distintos threads.

▸ **Método constructor:** Un constructor es un método inicializador. Se ejecuta al momento de instar una clase de modo automático. Su utilidad es la de inicializar valores de las variables miembro de la clase. Los constructores tienen el mismo nombre de la clase (de ese modo pueden ser identificados como constructores). No retornan ningún valor. Al igual que otros métodos, puede pasársele argumentos.

▸ **Recolector de basura:** En Java, ya no es necesario utilizar métodos destructores, ya que Java posee un "recolector de basura" que se encarga de eliminar de la memoria, por nosotros, aquellas variables que ya no serán utilizadas más adelante en el código.

CONTROL DE FLUJO

▸ **Bloques if:** Permite –en su versión simple- elegir si se ejecuta una sección de código (en el ejemplo, el código incluido en el "Bloque de sentencias"), dependiendo si se cumple una condición (definida en la expresión booleana). En su versión más completa, permite incluir además código a ejecutar si es que no se cumple la condición.

```
if(expresión booleana){
...
Bloque de Sentencias;
...
}
```

Otra manera es añadir if al final de los else:

```
if(expresión booleana){
...
Bloque de Sentencias;
...
}
else if(expresión booleana){
...
Bloque de Sentencias;
...
}
else {
...
Bloque de Sentencias;
...
}
```

▸ **Switch:** La sentencia *switch* proporciona una forma limpia de enviar la ejecución a partes diferentes del código en base al valor de una única variable o expresión. La expresión puede devolver cualquier tipo básico, y cada uno de los valores especificados en las sentencias *case* debe ser de un tipo compatible.

```
switch (expresión) {
 case n: sentencias;
 ...
 default: sentencias;
}
```

Una vez evaluada la expresión, se ejecuta la sentencia que sigue a la etiqueta *case*, y luego se continuan evaluando los *case* siguentes, y finalmente el *default*. Si fuera necesario, se debe agregar la sentencia *break* (usualmente al final de las sentencias que se ejecutan en cada *case*) para forzar una salida del bloque.

▸ **Bucle while:** Repite el código incluido en el "Bloque de sentencias" mientras "expresión booleana" sea verdadera. El código puede no ejecutarse nunca.

```
while(expresión booleana){
...
Bloque de Sentencias;

}
```

▸ **Bucle do-while:** Repite el código incluido en el "Bloque de sentencias" mientras "expresión booleana" sea verdadera. El código se ejecuta al menos una vez.

```
do {
...
Bloque de Sentencias;
...
} while (expresión booleana);
```

▸ **Bucle for:** Repite la ejecución del código incluido en el "Bloque de sentencias" mientras Condición sea verdadera (siguiendo el razonamiento anterior, el código puede no ejecutarse). En la primera vuelta ejecuta las Inicializaciones y revisa la Condición. Luego en cada vuelta ejecuta los Incrementos y vuelve a validar la Condición.

```
for (Inicializaciones;Condicion;Incrementos) {
...
Bloque de Sentencias;
...
}
```

▸ **Excepciones Try-catch:** Java implementa excepciones para apuntar al código robusto: cuando ocurre un error en un programa, el código lanza una excepción, que se puede capturar para tomar las medidas correctivas.

```
try {
 sentencias;
 }
catch (excepcion1) {
 sentencias;
 }
finally {
 sentencias;
 }
```

El cuerpo de la sección *try* se ejecuta hasta que finaliza con éxito o hasta que se lanza una excepción. Si sucede esto último, se busca entre los *catch* definidos para ver si se ha definido una excepción capturable. Por último se ejecuta la clausula finally, se haya producido o no una excepcion.

CLASES

Todas las acciones de los programas Java se colocan dentro del bloque de una clase. Una clase es la plantilla de la cual luego se crearán, o bien otras plantillas (herencia), o bien objetos (instancia).

▸ **Miembros de una clase**

Miembro	Definición
Campos	Variables de una clase. Pueden ser tipos primitivos como int, char, etc. O objetos.
Métodos	Código ejecutable, que define el comportamiento de los objetos.

▸ **Tipos de Clases**

Tipo	Definición
public	La clase es accesibles desde otras clases, bien sea directamente o por herencia.
abstract	La clase no se instancia, sino que se utiliza como clase base para la herencia. Las clases abstractas no pueden tener métodos privados puesto que no se podrían implementar, ni tampoco estáticos.
Final	La clase termina una cadena de herencia. No se puede heredar de esta clase final.

▸ **Nivel de acceso de los miembros de una clase**

Modificador	Definición
public	Miembros accesibles en cualquier lugar desde donde la clase sea accesible.
private	Miembros accesibles solo en la propia clase.
packaged	Accesibles en las clases del mismo paquete.
protected	Accesibles desde las subclases de la clase, en las clases del mismo paquete y, lógicamente, en la misma clase.
friendly	

▸ **Instancia de una clase y uso de un objeto:** Para crear un objeto instancia de una clase se utiliza el operador new:

Evaluacion objEvaluacion = new Evaluacion();

Al instanciar una clase, el objeto toma la funcionalidad definida en la clase (métodos), y las características inicializadas en el constructor.

Para llamar a alguna variable miembro o método del objeto instanciado se utiliza el separador "." (punto). Del siguiente modo:

objEvaluacion.calcularPromedio();

objEvaluacion.NotaMaxima;

this y super

En Java existen estas dos palabras reservadas, que nos ayudan a diferenciar las variables en un contexto adecuado. La palabra this se utiliza dentro de los métodos de una clase para hacer referencia a la clase misma y a sus miembros, del siguiente modo:

```
class volskwagen extends auto {
  int numeroPuertas = 2;

  void abrirPuerta(){
        int numeroPuertas = 4;
        numeroPuertas //tiene valor 4
        this.numeroPuertas //tiene valor 2
  }
}
```

La palabra super se utiliza para referenciar a la clase madre de la clase donde utilicemos dicha palabra.

▸ **Herencia:** Permite derivar una nueva clase (clase derivada) de una ya existente (clase base), la cual hereda los miembros de la clase base, tanto sus métodos como sus variables.

```
class volskwagen extends auto {
 int numeroPuertas = 2;
}
```

La palabra clave extends se usa para generar una subclase (especialización) de un objeto.

Se pueden sustituir los métodos proporcionados por la clase base. No existe la "herencia múltiple".

▸ **Polimorfismo:** Se dice que diferentes objetos son polimórficos si pueden responder –cada uno a su manera- a órdenes similares.

```
class cComputador{

  public void inicializar(int SO){
        ...
        Bloque de Sentencias;
  };
  public void inicializar(int SO, int modo){
        ...
        Bloque de Sentencias;
  };
  public void inicializar(){
        ...
        Bloque de Sentencias;
  };
}
```

▸ **Algunas clases importantes:** A continuación se describirán algunas de las clases más importantes para comenzar a programar. Una documentación más completa sobre cada clase y sus respectivos miembros se puede descargar de la página de Sun Microsystems. En la URL http ://java.sun.com/j2se/1.4.2/download.html, se puede encontrar el API del jdk versión 1.4.2.

▸ **Clase Math:** No se pueden crear instancias de la clase. Sin embargo, se pueden llamar desde cualquier sitio y no hay que inicializarla. Por ejemplo, para calcular el seno de un número: **rpta = Math.sin(0.5);**

Métodos	Comentarios
.sin(double)	Para int, long, float y double
.cos(double)	
.tan(double)	
.asin(double)	
.acos(double)	
.atan2(double,double)	

▸ **Clase Math:**

Métodos	Comentarios
.round(x)	Para double y flota.
.random()	Devuelve un double aleatorio.
.max(a,b)	Para int, long, float y double.
.min(a,b)	Para int, long, float y double
.E	Base exponencial.
.PI	Valor de PI.

▸ **Clases de variables numéricas:** Cada tipo numérico tiene su propia clase de objetos. Así el tipo float tiene el objeto Float. La primera sentencia creará una variable float (dato primitivo) y la segunda un objeto Float:

```
float f;
Float F;
```

Además existen algunas constantes propias de la clase. Así, un flotante tendrá las constantes:

```
Float.POSITIVE_INFINITY
Float.NEGATIVE_INFINITY
Float.NaN
Float.MAX_VALUE
Float.MIN_VALUE
```

Es posible pasar el valor de la variable de un tipo de dato o clase, a otro tipo de dato (conversión de datos) mediante los métodos propios de la clase.

```
String s = Float.toString( f );
f = Float.valueOf( "3.14" );
int i = F.intValue();
long l = F.longValue();
```

Existen métodos que nos ayudan a comprabar valores:

```
boolean b = Float.isNaN( f );
boolean b = Float.isInfinite( f );
```

isNaN() comprueba si *f* es un *No-Número,* por ejemplo, la raiz cuadrada de *-2.*

En los métodos *toString(), parseInt()* y *valueOf()* que no se especifica la base sobre la que se trabaja, se asume que es base 10.

▸ **Clase de variables String:** La clase *String* es un tipo de variable que representa una cadena de caracteres, pero que no cambia de valor una vez instanciada. Si necesitamos que dicha variable cambie de valor a lo largo del programa debemos usar el tipo de variable *StringBuffer*. Algunos de los métodos más básicos para el manejo de Strings, son:

Métodos	Descripción
int length	Devuelve la longitud de la cadena
char charAt(int indice)	Devuelve el caracter que se encuentra en la posición que se indica en indice.
boolean equals(Object obj)	Comparación de Strings.
Bolean equalsIgnoreCase(Object obj)	Lo mismo que equals() pero no tiene en cuenta mayúsculas o minúsculas
int compareTo(String str2)	Devuelve un entero menor que cero si la cadena es léxicamente menor que str2; cero si son iguales y un entero mayor que cero si es léxicamente mayor que str2.
boolean startsWith(String prefix)	Verifica si la cadena tiene un prefijo prefix.
boolean endsWith(String suffix);	Verifica si la cadena tiene un sufijo.
boolean startsWith(String prefix,int offset)	Verifica si la cadena tiene un prefijo prefix a partir de la posición offset.

▸ **DIRECTOR:** BERNARDO ROCA REY MIRÓ QUESADA ▸ **EDITORA:** FRANCA CAVASSA ▸ **EDITORA ADJUNTA:** MELISA OSORES ▸ **DESARROLLO:** MANUEL MASÍAS / LUIS DILLON ▸ **GERENTE COMERCIAL:** AUGUSTO TOWSEND ▸ **GERENTE DE PRODUCTO:** RENZO MARIÁTEGUI ▸ **JEFE DE PRODUCTO:** PAOLA CORNEJO ▸ **COORDINACIÓN:** DORA NIQUEN ▸ **EDITORA DE DISEÑO:** CLAUDIA BURGA-CISNEROS ▸ **DISEÑO:** VERUZKA NORIEGA ▸ **DIAGRAMACIÓN:** GERARDO CRISTÓBAL ▸ **EDICIÓN, PREPRENSA E IMPRESIÓN:** EMPRESA EDITORA EL COMERCIO S.A., PERÚ, AÑO 2004 TODOS LOS DERECHOS RESERVADOS. EMPRESA EDITORA EL COMERCIO S.A.